Choose a Job You Love: Predicting Choices of GitHub developers

Radoslaw Nielek*, Oskar Jarczyk*, Kamil Pawlak*, Leszek Bukowski*, Roman Bartusiak† and Adam Wierzbicki†
*Polish-Japanese Academy of Information Technology
86 Koszykowa Street, Warsaw, Poland
E-mail: {nielek, oskar.jarczyk, s3181, bqpro}@pja.edu.pl
†Wroclaw University of Technology
Wyb. Wyspianskiego 27, Wroclaw, Poland
E-mail: roman.bartusiak@pwr.edu.pl

***Abstract*—GitHub is one of the most commonly used web-based code repository hosting service. Majority of projects hosted on GitHub are really small but, on the other hand, developers spend most of their time working in medium to large repositories. Developers can freely join and leave projects following their current needs and interests. Based on real data collected from GitHub we have tried to predict which developer will join which project. A mix of carefully selected list of features and machine learning techniques let us achieve a precision of 0.886, in the best case scenario, where there is quite a long history of a user and a repository in the system. Even when proposed classifier faces a cold start problem, it delivers precision equal to 0.729 which is still acceptable for automatic recommendation of noteworthy projects for developers.**

***Index Terms*—Machine learning, link prediction, open-source software (OSS)**

I. Introduction

Complex projects can only be realized by teams. Single developer has neither enough time nor required skills to build big systems. It is even more obvious for open source projects, as majority of contributors work only part-time, and many of them do it in spare time, next to their professional duties. The motivations to join a particular project vary among developers. Some contributors want to fix a bug, others need a particular feature, yet another group use open source projects as a way to learn marketable skills or just to promote themselves. More details about motivations of open source developers can be find in Hertel et al. [1] and Roberts, Hahn et al. [2], [3].

Being able to predict who will join which project for such big communities like the GitHub opens completely new opportunities for project management. We could for example invite people even before they find project by themselves or predict which project will attract enough high quality developers and will eventually turn into a mature product. From the perspective of network science predicting of joining and leaving projects is an equivalent of link prediction problem. The main goal of this paper is to study whether it is possible to predict new links between the GitHub projects and developers with sufficient precision to make it usable in practice.

The requirement of practical applicability of proposed prediction algorithm enforces a very careful selection of features of projects/repositories and developers that have been used for making prediction. They have to be publicly available and easy to calculate. Moreover, as we plan to use it for recommending projects for developers (recommending means actually promoting) they have to be difficult (or expensive to manipulate) to make our recommendation system tamper-proof.

The remaining part of the paper is organized as follows. In chapter II a brief overview of previous works is presented. The chapter III discusses the GitHub dataset. In the following chapter some basic statistics related to the GitHub are shown. The fifth chapter contains a presentation of results. The last chapter concludes our paper and propose some directions for future works.

II. Related Work

One of the challenges in the field of Social Networks Analysis (SNA) is link prediction. Social networks are dynamic objects, which may grow or shrink over time. Some nodes may arrive while others depart. However, the same might be observed at the level of links, which connects nodes in a given network. The process of link formation might be even more dynamic, since new edges might connect nodes that are already present in the network and there is no need for new node arrival. As it was defined in [4] we study a basic computational problem underlying social-network evolution, the link-prediction problem: Given a snapshot of a social network at time t, we seek to accurately predict the edges that will be added to the network during the interval from time t to a given future time t'.

Generally speaking, there are two main divisions of methods used for link predictions: unsupervised and supervised. Unsupervised methods are mainly based on topology of the graphs [5] and especially on similarity metrics, which are in most of its merit variations about Common Neighbours of pair of nodes in the network. Most extensive overview of these unsupervised methods is in work [4]. Authors conduced experiments with several node similarity metrics (Graph Distance, Common Neighbours, Jaccard's Coefficient, Adamic/Adar Score, Preferential Attachment, Katz, Hitting

Time, Page Rank and Sim Rank). Their results suggest that Adamic Score [6] and Katz method (or its variants) are relatively close in their performance and provide good results. Indeed empirical data collected from Facebook in [7] shows that if a user creates an edge, then the probability that he links to a node with whom he has less than 4 friends is about 0.1%. Similar findings about link formation in the network of co-authorship and citation in scientific publishing are in [8]

In case of supervised methods [9], [10] one common approach consists of extracting network features from two nodes and combining them with node attributes, and using them in a supervised classifier. Backstrom and Leskovec [7] used Facebook data and proposed supervised random walks methodology. Their Predictions were made based on the scores of a random walk, but the edge weights for this walk were learned from node and edge attributes in a supervised way. The supervised random walk algorithm outperformed many other approaches, both supervised and unsupervised, on all datasets.

Link prediction on bipartite networks is a much less explored topic than general link prediction. However, in [11] authors proposes an algorithm to cope with the link prediction problem on bipartite networks. They describe a temporal bipartite projection method that yields a projected item graph, called the temporal projection graph (TPG). Based on the TPG, they propose a scoring function called STEP (Score for TEmporal Prediction) for each user-item pair. STEP leverages the historical behaviours of individual users and the social aggregated behaviours learned from the TPG for the link prediction problem. Several experiments are performed on DBLP author-conference dataset, the Flickr dataset and the Delicious dataset. In [12] authors copes with the problem of link prediction in large-scale two-mode social networks. They convert one side of the bipartite network into a single network with homogeneous nodes and then they apply similarity measures. These similarity score is turned into node feature and used in a supervised link prediction classifier. Experimental validation of the proposed approach is carried out on two real data sets: a co-authoring network extracted from the DBLP bibliographical database and bipartite graph history of transactions on an on-line music e-commerce site. Link prediction on bipartite networks is also strongly connected with recommendation engines [13], [14]and especially with collaborative filtering methods. For example in [15] transactions are mapped to a bipartite user–item interaction graph and recommendation problem is converted into a link prediction problem. Authors propose a kernel-based approach to link prediction and recommendation. They design a graph kernel to exploit features in the context of focal user–item pair. The kernel works with a one-class SVM algorithm to predict user–item interactions. They prove the validity and computational efficiency of the graph kernel.

Benefits coming from the application of recommendation systems to software engineering have been obvious for researchers for many years. Early recommendation systems were focused on increasing productivity of developers working with big online repositories of code. Happel and Maalej [16] published a overview of early application of recommendation systems to increase productivity of software engineers. Further studies on similar topic can be found in [17]. Rapid development of online repositories of open source projects containing millions of projects and billions lines of code forces researchers to look for innovative ways to explore it. McMillan et al. [18] proposed a variety of methods to measure software similarities.

Selection of team members is the most important factor influencing success of open source projects and, at the same time, the most frequently studied. Social capital and structural holes in collaboration networks have been researched by [19], Singh et al. [20] focused on small-world network properties. Extensive review of factors influencing probability of success for open source projects can be found in [21] and [22].

Zhang et al. [23] published interesting exploratory study about possibility of recommending relevant projects on GitHub based on similarities in behaviours between users. Although they discovered interesting patterns they do not propose and evaluate a recommendation system. Casalnuovo et al. [24] have studied importance of prior social links in decision to join a GitHub repository. It seams that developers prefer projects where they have some pre-existing relationships to people who are already members. On the other hand, all conclusions about developers behaviours drawn only from GitHub data may be skewed as Wu et al. shows that substantial part of interaction between developers active on GitHub take place outside this web site [25] and, thus, cannot be easily traced.

Next to variables related to projects and developers that can be measured directly researchers proposed also more sophisticated methods for extracting knowledge from GitHub repositories. Jarczyk et al. [26] suggested methods for estimating projects quality based on project popularity and support offered by team members to users. Restricted Boltzmann Machine was used by Bartusiak et al. [27] for predicting cooperation between developers. Some inspiring approaches can also be found in papers studying cooperation in other crowdsourcing systems – e.g. Jankowski et al. [28] constructed a knowledge network for Wikipedia editors based on their history of edits. Collaboration on the Wikipedia has been extensively studied [29], [30], [31].

Feasibility of even the most advanced recommendation algorithm may be limited by required computation that has to be done. Deja et al. [32] show that even for huge number of objects an viable approximate solution may be proposed.

III. Dataset

Our dataset is based on GHTorrent - a scalable and queriable offline mirror of data offered through the GitHub REST API. For each GitHub public event, GHTorrent retrieves its contents and dependencies. It then stores the raw JSON responses to a MongoDB database, while also extracting their structure to a MySQL database. Since 2012 for every two months GHTorrent has released data which was collected during that period of time. The relational data schema and basic description of the data base might be found here: http://ghtorrent.org/relational.htm

For the purposes of our research, we have used GHTorrent data from August 2014. We have chosen projects that were at least two years old (as of 09.2014), with at least 100 commits and at least 5 individual (non organization) team members. The resulting dataset consists of 9447 GitHub projects and 62607 users. The data is then split into separate quarters, starting with Q1 2011 and ending with Q2 2014.

Following that, we analyse developers' programming activity within repositories, as measured by number of commits within project. We only take into account users that are formal project members at the time of a particular commit. In particular, we look for a situation where developer has commits during N quarter, but no commits during previous $N-1$ quarter. This shows that a new connection has appeared during that time.

After that we created randomized developer-project pairs which had no commits during either of those quarters. Developers were chosen from all users that had already created GitHub account during that quarter. The number of negative examples (such developer - repository pairs that link between them did not appear) is equal to the number of positive examples (developer - repository pairs where new link were created) in that particular quarter.

Over 50 distinct features have been constructed and calculated for developers and repositories. The selection of the most important features for developers and repositories is presented in tab. I and tab. II respectively. All variables have been calculated for $N-1$ quarter.

List of languages used by developers (*D_languages*) and list of languages that are related to the repository (*R_languages*) have been used to calculate skills similarity (*similarity*) between developer and repository. Similarity is expressed as cardinality of intersection of *R_language* and *D_language* divided by cardinality of a sum of *R_language* and *D_language*.

IV. Basic statistics

This chapter presents some summaries of the dataset, with emphasis the dynamics of how often new software developers joined repositories, as well as some plots showing distribution of team members. Figure no. 1 shows average number of new developers which joined a repository in a particular month (counted from month 0, which is the first month repository was created).

Plot shows two noticeable peaks in value, which are moments of high income of new team members. First one, is the beginning of the project, which is quite obvious, because it may correspond to the very first moment of a) formalization of the software team, which existed outside GitHub; or b) when team members are appointed by a repository administrator to users known to him/her. Next, the average number of team joins shows a positive trend towards the second peak (around the 70th month), which may represent the power of mature and well-know repositories to attract new software developers.

Name	Description
D_degree	*number of projects* joined by a developer
D_languages	*list of programming languages* in which a developer made code commits
D_gender	*developer's gender* (calculated basing on a name, male by default)
D_followers	how many *followers* a developer has?
D_following	*how many users* are *followed* by a developer?
D_exp_in_days	how many *days* developer's GitHub account exists?
D_repo_starred	did a developer award this repository a *star*?
D_repo_discussion	how many *comments* a developer made on project commits and issues?
D_coowork_repo	how many developers, that *co-work* in other projects (together with this developer), already joined this repository?
D_owned_repos	number of projects owned by a developer
D_owned_stars	number of *stars* in developer's owned projects
D_owned_forks	number of *forks* in developer's owned projects
D_owned_issues	number of *issues* in developer's owned projects
D_reported_issues	number of *issues reported* by a developer
D_assigned_issues	number of *issues assigned* to a developer

TABLE I: Variables related to a developer

Name	Description
R_degree	number of developers that *joined* this repository
R_languages	*programming languages* used in this project
R_stars	numbers of *stars* given to this repository
R_is_fork	is this project a *fork* of another repository?
R_commits	number of *commits* made to the project in this quarter
R_members	number of *project members*
R_forks	number of *forks* of the project
R_committers	number of active *committers* in the project

TABLE II: Variables related to the repository

Kalliamvakou et al. in [33] describes two models of a team membership, one is through a code commit (a 'pull request' or a direct 'push' of one or more commits), while second is a project membership through a direct indication by the repository owner or administrator (a feature in GitHub called 'teams').

Let's first see an overall statistics of team size on the *whole GitHub Torrent* database. Figures no. 2 and no. 3 show a distribution of the repository size, where project size is understood as the *number of committers* to the particular repository. Second figure shows the same value but after logarithmic transformation, to better show the distribution, which is very skewed and decline exponentially.

Both plots confirm some of the findings by [33], which states that most of the projects consists of small teams (more than $7.5m$ projects are of size between 0 and 10 committers) or are 'private repositories'. Some of the repositories do not have any commits at all and are empty, this situation is possible on the GitHub portal, which offers the possibility of disabling creation of 'README.md' file.

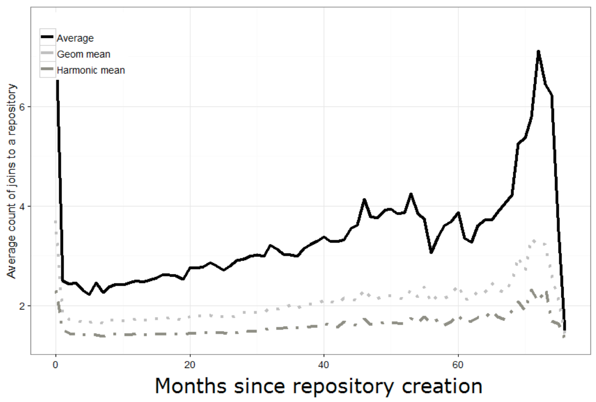

Fig. 1: New joins (in average) since repository was created

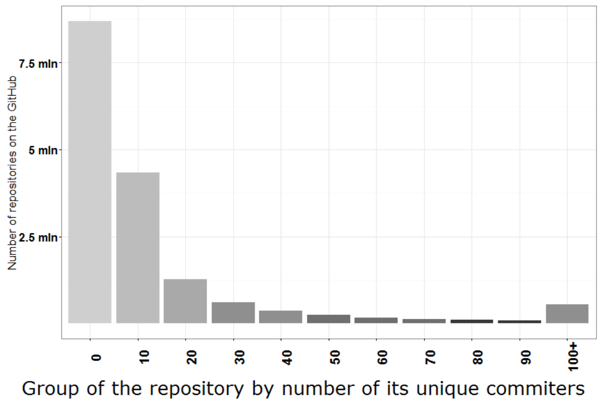

Fig. 2: GitHub, repository size by committers

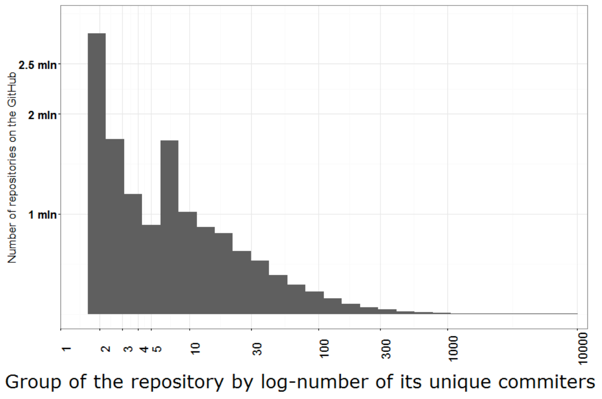

Fig. 3: Repository size after log transform

Table no. III concern the same global view, but shows a distribution of the repository size, in the model where team members are developers assigned by the repository owner and/or administrator.

The distribution is even more uneven, which means that the 'team' functionality is much less popular than a traditional 'pull request' workflow. Moreover, a repository owner can be treated as a team member (asterisked note in table no.III),

Members range	**Count**	**Density**
$[0, 10)$*	25607437	0.0999
* *includes repository with no members but owner*		
$(0, 10)$	4264188	0.0994
$[10, 20)$	16994	0.0004
$[20, 30)$	5126	0.0001
$[30, 40)$	3607	0.00005
40+	5151	0.001

TABLE III: GitHub, repository size by team members

despite the fact that the GH Torrent originally does not include them, due to data provided by the GitHub API itself.

Let's now focus on the sample of selected repositories described in the 'Dataset' paragraph no. III. Figure no. 4 show a distribution of the repository number of project members (defined by both, 'committers' and implicit 'team members').

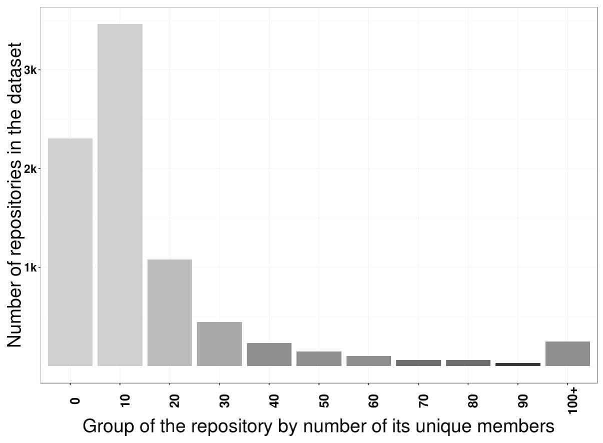

Fig. 4: Dataset sample, number of members

Plot shows a significant increase in the bucket of team size between 10 and 20 members, which is a promising characteristic for our dataset and means that both small and medium sized teams prevail within selected repositories.

Further analysis show that around $1/3$ of repositories in the dataset have more than 500 commits, as seen on table no. IV.

Commit count	**No of repositories**	**Density**
$[450, 460)$	64	0.00101
$[460, 470)$	57	0.00090
$[470, 480)$	57	0.00090
$[480, 490)$	41	0.00065
$[490, 500)$	29	0.00046
500+	3145	0.000005

TABLE IV: Dataset, number of commits in a repository

V. Results

A. Prediction

Based on calculated features of developers and repositories for the quarter *N*, that are described in section III, we wanted to predict which developer will join which repository in the next quarter. We have mixed different quarters in our dataset to avoid overfitting to one particular point in time. Moreover, we

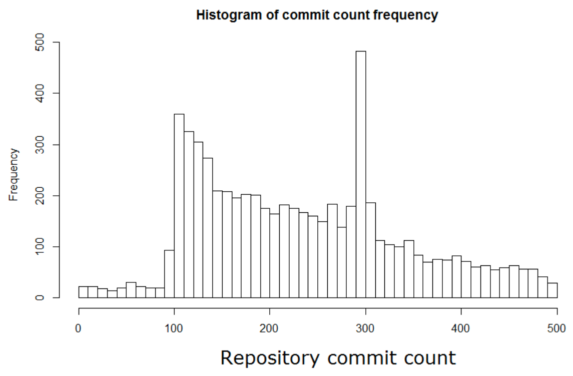

Fig. 5: Dataset sample, histogram of number of commits

wanted to prove that discovered regularities can be generalized beyond single point in time. Classifiers appear to perform well for all quarters despite the evolution of the whole service (and community).

We divided our dataset to two parts: training dataset that contains 8738 repository-developer pairs (randomly selected 10% of all pairs) and testing dataset that contains remaining 78436 objects. As a classifier we used a random forest algorithm from the "randomForest" package in version 4.6-12 installed in R version 3.2.3. We decided for 50 trees (we tested different parameters but without much changes in results). As there is the same number of positive and negative cases the benchmark based on random assignment to one of classes is equal to 0.5 for precision and recall.

Following subsections present results for different subsets of features. As we could expect removing some features from the dataset make the task of predicting link harder but also reveals the real importance of particular features. Results for all available features, that are at the same time the best results we achieved, are presented in section V-B.

B. All available features

Results for classifier learned on all available features are presented in table V. The precision peaks at 0.888 which is surprisingly high results for such complex task. Performance of a classifier is quite stable for all quarters with only a minor variation that can be explained by stochastic nature of the learning process.

The most important features were: *D_degree*, *R_degree* and *R_commits*. It seems that popular projects are more frequently selected by developers. Random forest classifier only returns information about variables importance and not about the nature of this relationship. We decided to take closer look on variable *R_degree* to verify this hypotheses.

We selected a particular point in time (Q3 2013[1]) and split all projects into two groups: first group contains all projects that will not have new members in the next quarter and second group contains only projects that attract users in the next quarter. The average number of project members for projects belonging to the first group is lower than for projects from the second group – 1.65 and 5.63 respectively.

[1]The results for other quarters are similar.

	0	1	
0	35958	5334	87,08%
1	3452	33858	90,75%
	91,24%	86,39%	

TABLE V: Classification results for all available features. Precision=0.888, recall=0.864

The difference between two groups mentioned in the previous paragraph is quite big. Part of explanation is that many projects on the GitHub have been created only for personal use and are not intended to be shared with anyone. Therefore, we decided to take a closer look at projects that survived at least two years, and have at least 100 commits, and at least 5 individual team members. There are 9447 such projects which are subscribed by 62607 users. For such restricted case we can still observe a difference between averages, albeit much lower. The average number of members is 8.417 and 9.092 respectively. The sign of preferential attachment strategy can be also observed on fig. 6. The log-log histogram of number of repository members depicted there a resembling a power-law like distribution.

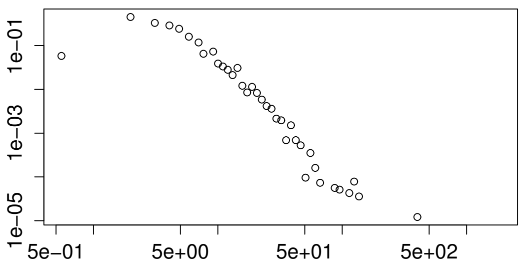

Fig. 6: Log-log histogram of number of repository members

C. Without information about repository degree and developer degree

Two variables: *D_degree* and *R_degree* are by far the most important in predicting which developers join which repository. On one hand, it is convenient because we can limit the number of features we calculate; on the other hand, it makes any attempts to manipulate recommendation algorithm quite easy by adding fake users to repository. Therefore we remove these variables and try to learn classifier by on remaining features. As could be expected, the performance of the classifier are worse but still acceptable – precision reached 0.827. Detailed results are presented in table VI. The most important variables are: *R_commits* followed by *D_exp_in_days* and *R_stars*.

	0	1	
0	32023	6235	83.70%
1	7296	32843	81.82%
	81.44%	84,04%	

TABLE VI: Classification results for all features but *R_degree* and *D_degree*. Precision=0.827, recall=0.840

	0	1	
0	25972	18774	58,04%
1	13381	20309	60,28%
	66,00%	51,96%	

TABLE VII: Results of classification based only on *similarity* between developers and repositories. Precision=0.590, recall=0.520

	0	1	
0	31877	13777	69,82%
1	7437	25332	77,30%
	81,08%	64,77%	

TABLE VIII: Results of classification based *similarity* and *R_degree*. Precision=0.729, recall=0.648

	0	1	
0	25735	13259	66,00%
1	13565	25894	65,62%
	65,48%	66,14%	

TABLE IX: Results of classification based *similarity* and *R_coowork_repo*. Precision=0.658, recall=0.661

D. Prediction based on similarity between developers and repositories

Features used in previous sections to predict link between repository and developer will be prone for cold start problem. For freshly created repositories or newly registered users majority of variables presented in table II and I will be zero. The problem is not only with the lack of data but also in that there is no way to speed up the process of generating data for new repositories and users. People must grow up in community to collect real followers, report or close issues and take part in discussions. Actually, one of very few questions we can ask developers (and those who create repository) and expect meaningful answers is: which programming languages do you use/which programming languages are required for this project.

In the first experiment described in this section we wanted to check whether such information is sufficient to accurately recommend repository to developer. We calculated *similarity* between languages mastered by developer and those which are required for a given project. The exact method is described in section III. After that we trained classifier based only this single feature. Results are summed up in table VII. Precision and recall are higher than for random assignment benchmark but still very low – 0.59 and 0.52 respectively. So, the hardest scenario where both the programmer and the repository do not have any history in GitHub cannot be successfully addressed within our framework.

The scenario presented in previous paragraph is not only the most difficult but also relatively uncommon and, thus it has limited practical importance. More commonly we will have influx of new users that we have to match with the most interesting repositories that already exists in GitHub for some time. To model such situation we have tried to recommend links based only on *similarity* and *R_degree*. Obtained results are much better than for previous scenario and reached 0.729 for precision and 0.648 for recall. Confusion matrix is presented in table VIII.

Yet another option is that we have mature developers that already worked in different projects and newly created repository. We can measure similarity between repository and developer but not much more. At least at the beginning we cannot expect that there will be a substantial number of developers that joined the repository (low *R_degree*), virtually no commits (*R_commits*) and definitely no stars (*R_stars*).

Our hypothesis was that the very first developers that join repository right after it has been created can attract their acquaintances from other projects. If such effect appears, then even a quite small number of developers that joined a project may allow a precise recommendation. Table IX shows the results of prediction for classifier learned on only two features: *similarity* and *R_coowork_repo*. Although it preforms better than recommendation based only on *similarity* the results are pretty low.

VI. Discussion

Precise recommendation is not an easy task and it is even harder when the decisions taken by users are expensive in terms of time, money, or more general resources. Buying a book, watching a video clip or even making new friendships on the social network site is much less demanding than joining a open source project and contribute to it. Moreover, developers can use almost all functions of GitHub without joining a project. Nevertheless, obtained results indicate that it is completely viable to build a recommendation system for GitHub that performs on acceptable level.

Our study has partially confirmed the findings published by Casalnuovo et al. [24] that having friends working in repository is one of factors influencing developers' decision to join a particular project. This single feature is strong (albeit not the strongest) prediction of developers' behaviour but alone is not enough. Almost all features proposed in table I and table II play some role in prediction and, at the same time, helps to limit the influence of data inconsistency or purposeful manipulation.

Precision at 0.886 for the best case scenario and 0.729 for new users is impressive for such complex task and makes direct use of proposed classifier in real system entirely feasible. Small improvements in precision and recall are still possible by tweaking the parameters of used random forest algorithm. In the next step we want to implement proposed recommendation algorithm as a browser plug-in and make it publicly available.

Incorporating external to GitHub data sources of developers behaviours and repositories popularity is the most promising direction for future study. Popularity of projects and technologies in social media and developers' social networks acquired from on-line forums (e.g. Stack Overflow[2]) might help us to make better recommendation. Another possibility would be the creation of a reputation system [34], [35], [36] for GitHub developers.

Further study is also required to evaluate the influence of project recommendation system on the whole community. There is a plenty of open source projects that address all problems we can imagine, partly because of a huge fragmentation and long tail effect. Recommendation system might cause that "rich get richer" and thus less projects would have a chance to sustain for a longer period of time. Yet another problem for recommendation system is to differentiate developers according to their motivations. Some developers join project because they want to learn something, others want to solve well-defined problems, yet another group looks for recognition and fame. Therefore, one size fits all approach will never work and a motivation recognition mechanism should be embedded into recommendation system.

Acknowledgments

This research is partially supported by the Polish National Science Centre grant 2012/05/B/ST6/03364 and the European Unions Seventh Framework Programme for research, technological development and demonstration under grant agreement no 316097[ENGINE].

References

[1] G. Hertel, S. Niedner, and S. Herrmann, "Motivation of software developers in open source projects: an internet-based survey of contributors to the linux kernel," *Research policy*, vol. 32, no. 7, pp. 1159–1177, 2003.

[2] I.-H. Hann, J. Roberts, S. Slaughter, and R. Fielding, "Why do developers contribute to open source projects? first evidence of economic incentives," in *2nd Workshop on Open Source Software Engineering, Orlando, FL*, 2002.

[3] J. A. Roberts, I.-H. Hann, and S. A. Slaughter, "Understanding the motivations, participation, and performance of open source software developers: A longitudinal study of the apache projects," *Management science*, vol. 52, no. 7, pp. 984–999, 2006.

[4] D. Liben-Nowell and J. Kleinberg, "The link prediction problem for social networks," in *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, ser. CIKM '03. New York, NY, USA: ACM, 2003, pp. 556–559. [Online]. Available: http://doi.acm.org/10.1145/956863.956972

[5] Z. Huang, "Link prediction based on graph topology: The predictive value of generalized clustering coefficient," *Available at SSRN 1634014*, 2010.

[6] L. A. Adamic and E. Adar, "Friends and neighbors on the web," *Social networks*, vol. 25, no. 3, pp. 211–230, 2003.

[7] L. Backstrom and J. Leskovec, "Supervised random walks: Predicting and recommending links in social networks," in *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, ser. WSDM '11. New York, NY, USA: ACM, 2011, pp. 635–644. [Online]. Available: http://doi.acm.org/10.1145/1935826.1935914

[8] T. Martin, B. Ball, B. Karrer, and M. E. J. Newman, "Coauthorship and citation in scientific publishing," *CoRR*, vol. abs/1304.0473, 2013. [Online]. Available: http://arxiv.org/abs/1304.0473

[2] http://stackoverflow.com

[9] N. Benchettara, R. Kanawati, and C. Rouveirol, "A supervised machine learning link prediction approach for academic collaboration recommendation," in *Proceedings of the fourth ACM conference on Recommender systems*. ACM, 2010, pp. 253–256.

[10] M. Al Hasan, V. Chaoji, S. Salem, and M. Zaki, "Link prediction using supervised learning," in *SDM06: Workshop on Link Analysis, Counterterrorism and Security*, 2006.

[11] T. Wu, S.-H. Yu, W. Liao, and C.-S. Chang, "Temporal bipartite projection and link prediction for online social networks," in *Big Data (Big Data), 2014 IEEE International Conference on*, Oct 2014, pp. 52–59.

[12] N. Benchettara, R. Kanawati, and C. Rouveirol, "Supervised machine learning applied to link prediction in bipartite social networks," in *Advances in Social Networks Analysis and Mining (ASONAM), 2010 International Conference on*, Aug 2010, pp. 326–330.

[13] A. Giel, S. Jain, and K. Wang, "Group 14 predicting yelp reviews using internal links - link prediction in bipartite networks," 2013.

[14] O. Allali, C. Magnien, and M. Latapy, "Link prediction in bipartite graphs using internal links and weighted projection," in *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on*, April 2011, pp. 936–941.

[15] X. Li and H. Chen, "Recommendation as link prediction in bipartite graphs: A graph kernel-based machine learning approach," *Decision Support Systems*, vol. 54, no. 2, pp. 880 – 890, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167923612002540

[16] H.-J. Happel and W. Maalej, "Potentials and challenges of recommendation systems for software development," in *Proceedings of the 2008 international workshop on Recommendation systems for software engineering*. ACM, 2008, pp. 11–15.

[17] M. P. Robillard, R. J. Walker, and T. Zimmermann, "Recommendation systems for software engineering," *Software, IEEE*, vol. 27, no. 4, pp. 80–86, 2010.

[18] C. McMillan, M. Grechanik, and D. Poshyvanyk, "Detecting similar software applications," in *Software Engineering (ICSE), 2012 34th International Conference on*. IEEE, 2012, pp. 364–374.

[19] Y. Tan, V. Mookerjee, and P. Singh, "Social capital, structural holes and team composition: Collaborative networks of the open source software community," *ICIS 2007 Proceedings*, p. 155, 2007.

[20] P. V. Singh, "The small-world effect: The influence of macro-level properties of developer collaboration networks on open-source project success," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 20, no. 2, p. 6, 2010.

[21] J. Wang, M. Y. Hu, and M. Shanker, "Human agency, social networks, and foss project success," *Journal of Business Research*, vol. 65, no. 7, pp. 977–984, 2012.

[22] V. Midha and P. Palvia, "Factors affecting the success of open source software," *Journal of Systems and Software*, vol. 85, no. 4, pp. 895–905, 2012.

[23] L. Zhang, Y. Zou, B. Xie, and Z. Zhu, "Recommending relevant projects via user behaviour: An exploratory study on github," in *Proceedings of the 1st International Workshop on Crowd-based Software Development Methods and Technologies*, ser. CrowdSoft 2014. New York, NY, USA: ACM, 2014, pp. 25–30. [Online]. Available: http://doi.acm.org/10.1145/2666539.2666570

[24] C. Casalnuovo, B. Vasilescu, P. Devanbu, and V. Filkov, "Developer onboarding in github: The role of prior social links and language experience," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: ACM, 2015, pp. 817–828. [Online]. Available: http://doi.acm.org/10.1145/2786805.2786854

[25] Y. Wu, J. Kropczynski, P. C. Shih, and J. M. Carroll, "Exploring the ecosystem of software developers on github and other platforms," in *Proceedings of the Companion Publication of the 17th ACM Conference on Computer Supported Cooperative Work &#38; Social Computing*, ser. CSCW Companion '14. New York, NY, USA: ACM, 2014, pp. 265–268. [Online]. Available: http://doi.acm.org/10.1145/2556420.2556483

[26] O. Jarczyk, B. Gruszka, S. Jaroszewicz, L. Bukowski, and A. Wierzbicki, "Github projects. quality analysis of open-source software," in *Social Informatics*. Springer International Publishing, 2014, pp. 80–94.

[27] R. Bartusiak, T. Kajdanowicz, A. Wierzbicki, L. Bukowski, O. Jarczyk, and K. Pawlak, "Cooperation prediction in github developers network with restricted boltzmann machine," in *Intelligent Information and Database Systems*. Springer Berlin Heidelberg, 2016, pp. 96–107.

[28] M. Jankowski-Lorek, S. Jaroszewicz, L. Ostrowski, and A. Wierzbicki, “Verifying social network models of wikipedia knowledge community,” *Information Sciences*, vol. 339, pp. 158 – 174, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025515009044

[29] A. Wierzbicki, P. Turek, and R. Nielek, “Learning about team collaboration from wikipedia edit history,” in *Proceedings of the 6th International Symposium on Wikis and Open Collaboration*. ACM, 2010, p. 27.

[30] P. Turek, A. Wierzbicki, R. Nielek, A. Hupa, and A. Datta, “Learning about the quality of teamwork from wikiteams,” in *Social Computing (SocialCom), 2010 IEEE Second International Conference on*. IEEE, 2010, pp. 17–24.

[31] P. Turek, A. Wierzbicki, R. Nielek, and A. Datta, “Wikiteams: How do they achieve success?” *IEEE Potentials*, vol. 30, no. 5, pp. 15–20, 2011.

[32] D. Deja, R. Nielek, X. Lin, and A. Wierzbicki, “Hybrid algorithm for precise recommendation from almost infinite set of websites,” in *2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT), Warsaw, Poland, August 11-14, 2014 - Volume II*, 2014, pp. 318–322. [Online]. Available: http://dx.doi.org/10.1109/WI-IAT.2014.50

[33] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, “The promises and perils of mining github,” in *Proceedings of the 11th working conference on mining software repositories*. ACM, 2014, pp. 92–101.

[34] T. Kaszuba, A. Hupa, and A. Wierzbicki, “Advanced feedback management for internet auction reputation systems,” *IEEE Internet Computing*, vol. 14, no. 5, pp. 31–37, 2010.

[35] R. Nielek, A. Wawer, and A. Wierzbicki, “Spiral of hatred: social effects in internet auctions. between informativity and emotion,” *Electronic Commerce Research*, vol. 10, no. 3-4, pp. 313–330, 2010.

[36] A. Wierzbicki, “The case for fairness of trust management,” *Electronic Notes in Theoretical Computer Science*, vol. 197, no. 2, pp. 73–89, 2008.